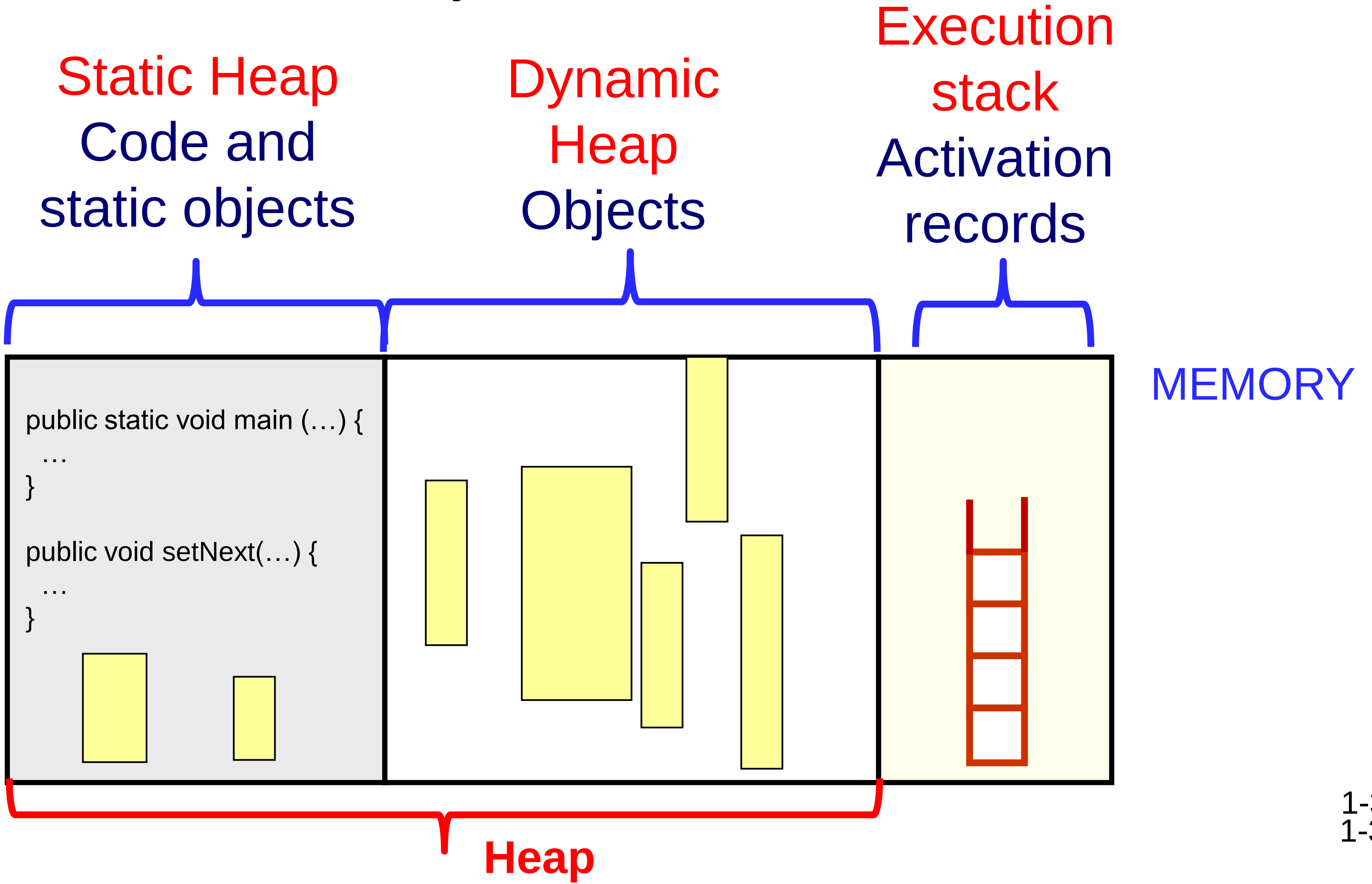
Memory Management

Objectives

- Understand how the computer's memory is used when executing a Java program
- Identify the different parts of memory for storing classes, objects, and the execution stack

Memory Allocation in Java

- When a program is being executed, separate areas of memory are allocated:



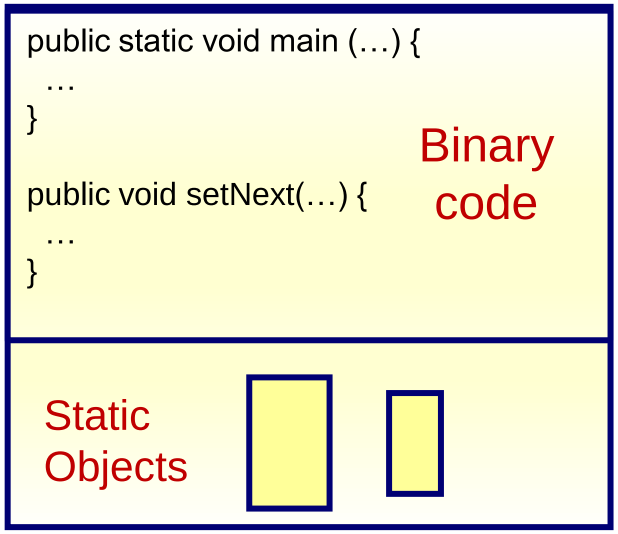
1-3

Static Heap

Used to store

- code for class methods
- static objects

The amount of memory used for this area is fixed (does not change) during the execution of a program, because the code does not change, and new static objects cannot be created.



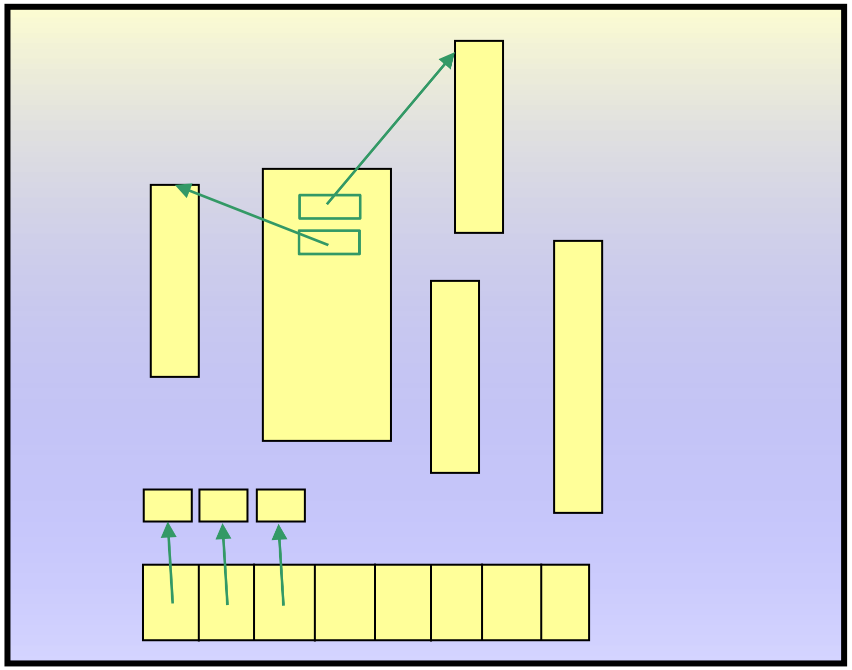
```
public static void main (…) {
  …
}

public void setNext(…) {
  …
}
```

Dynamic Heap

Used to store objects dynamically created during the execution of a program. Information that is stored for each object:

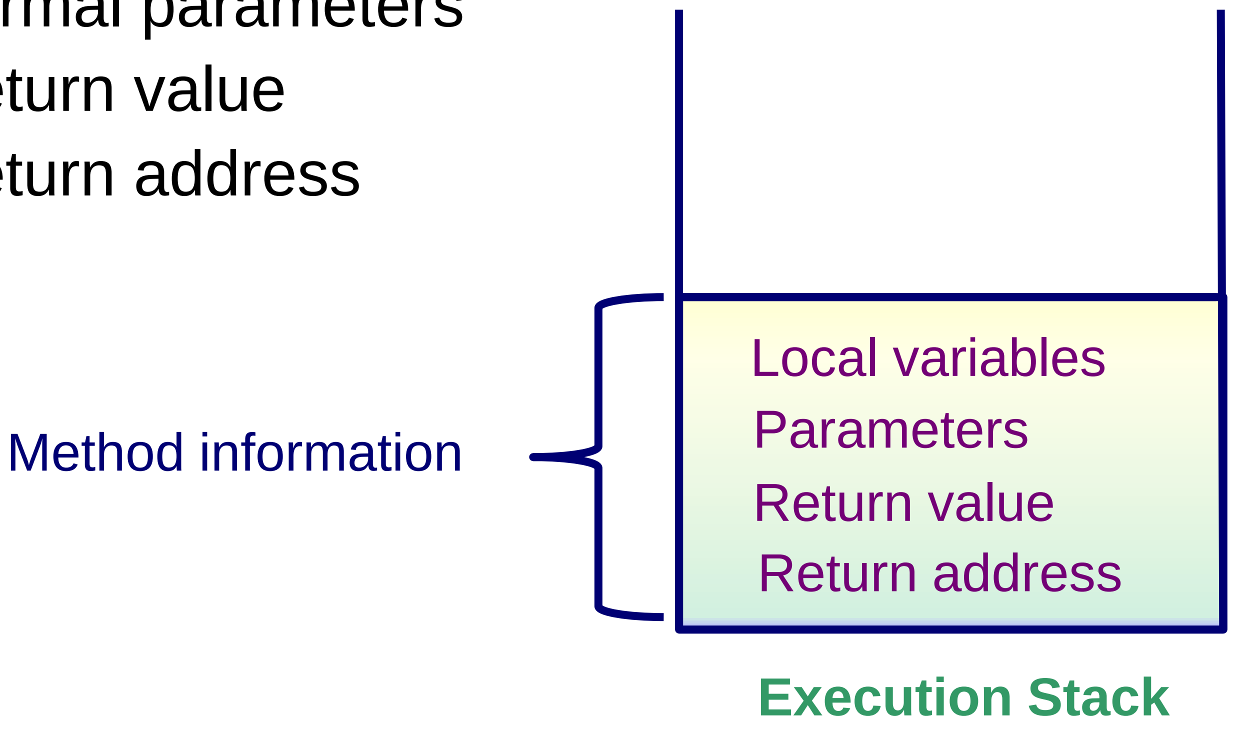
- values of its instance variables
- reference to its code



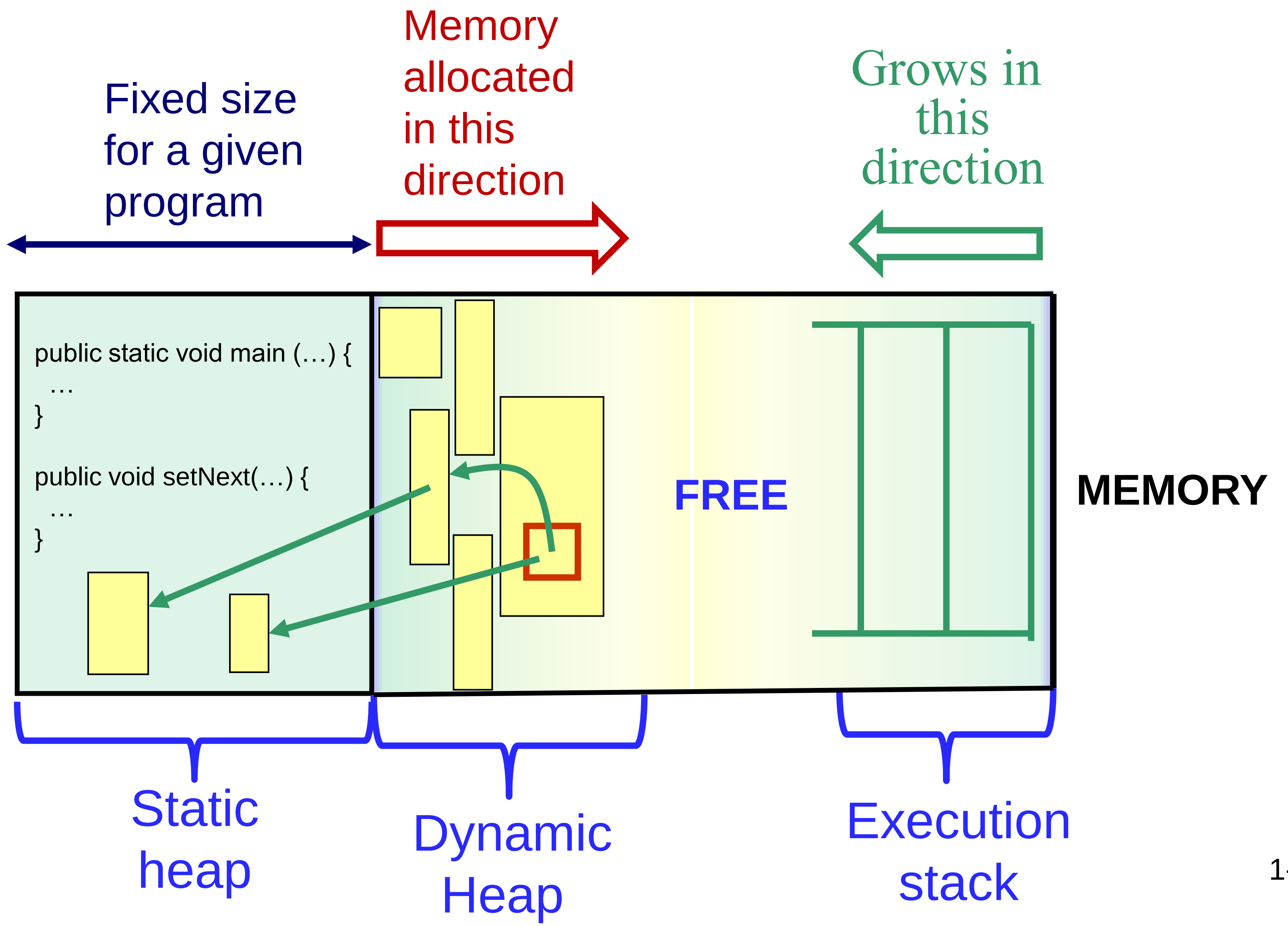
Execution Stack

The execution stack (also called runtime stack or call stack) is used to store information needed while a method is being executed, like

- Local variables
- Formal parameters
- Return value
- Return address



Memory Allocation for a Program

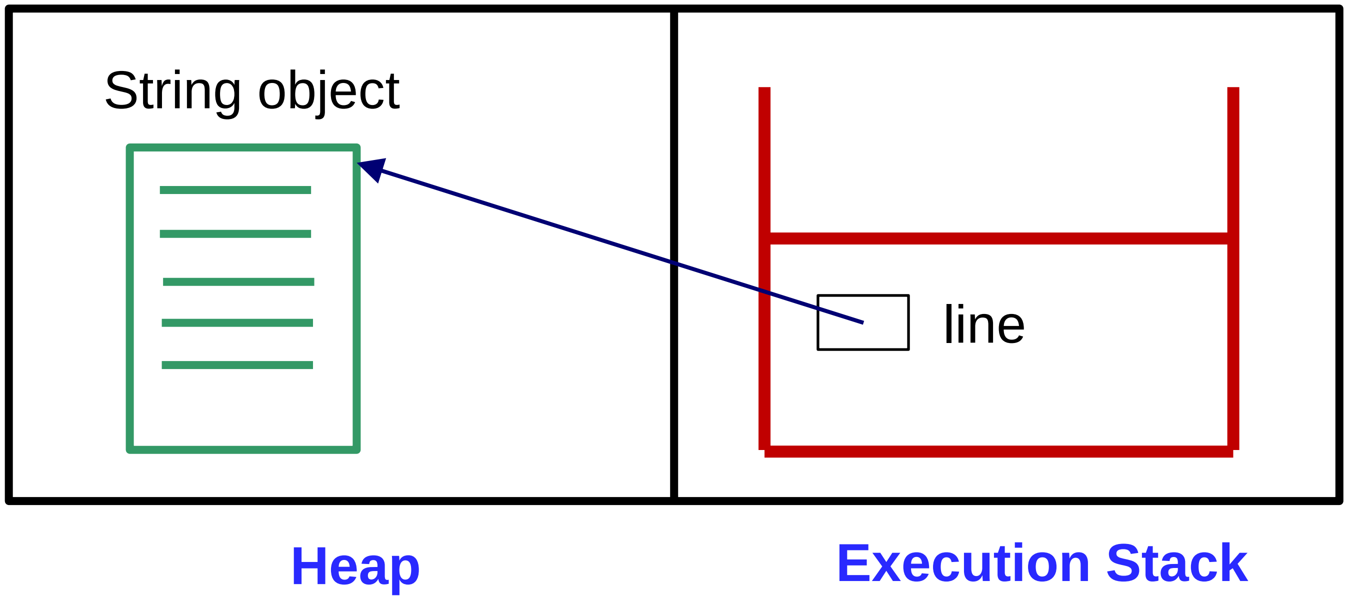


Memory Allocation in Java

What happens when an object is created in a method by the new operator, as in

String line = new String("hello");?

- The local variable line has memory allocated to it in the execution stack
- The object is created in the heap



Execution Stack

- Execution stack (or runtime stack or call stack) is the memory space used to store the information needed by a method, *while the method is being executed*
- When a method is invoked, an activation record (call frame, stack frame, or frame) for that method is created and pushed onto the execution stack
  - All the information needed during the execution of the method is stored in its activation record

Activation Record

- An activation record contains:
  - Address to return to after method ends
  - Method’s parameters
  - Method’s local variables
  - Return value (if any)

- Note that the values stored in an activation record are accessible only while the corresponding method is being executed!

How Programs are Executed

Consider the following Java program.

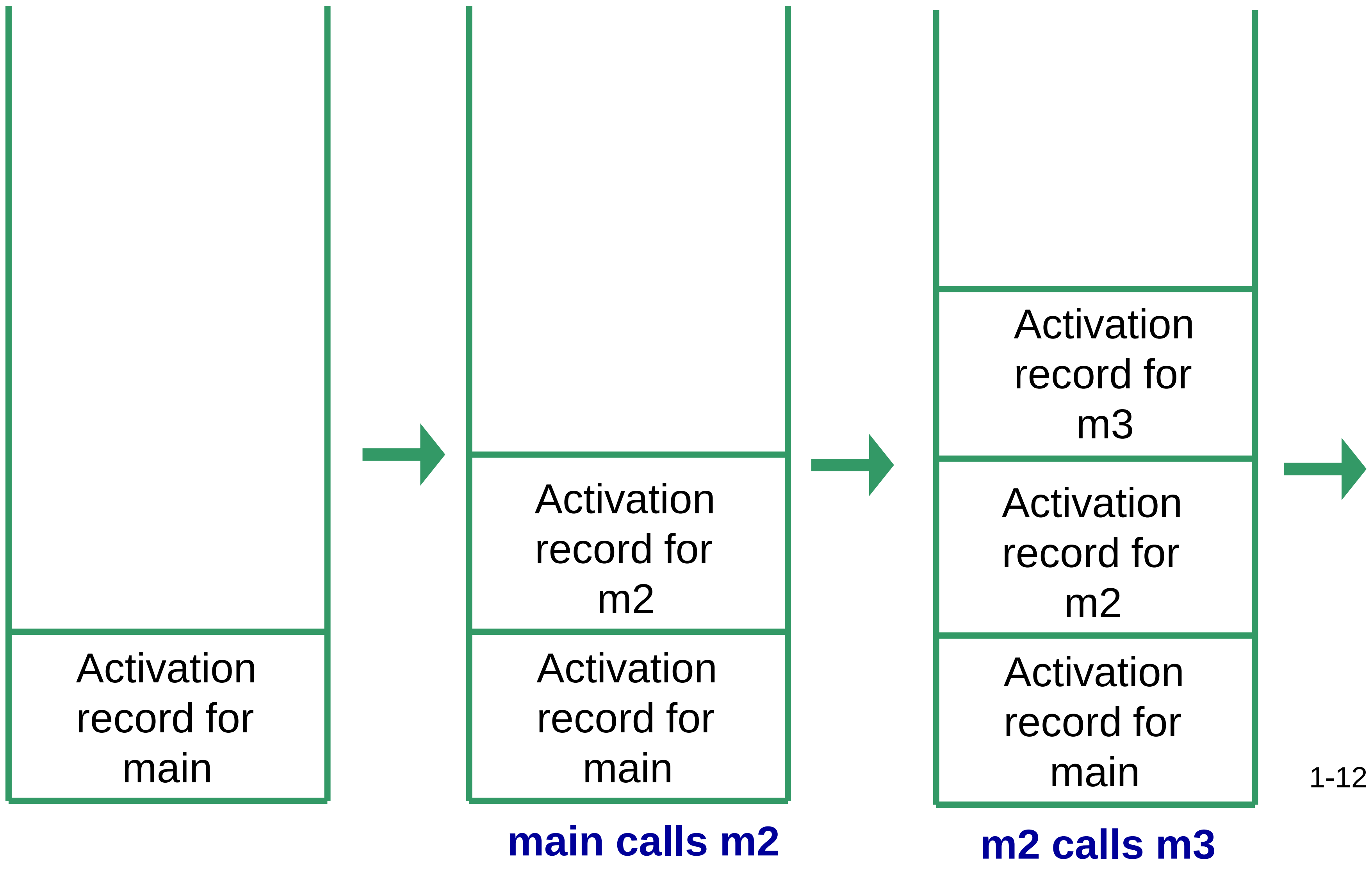
```
public static void m4( ) {
    System.out.println("Starting m4");
    System.out.println("Leaving m4");
    return;
}

public static void main(String args[ ]) {
    System.out.println("Starting main");
    System.out.println("main calling
    m2");
    m2( );
    System.out.println("Leaving main");
}
```

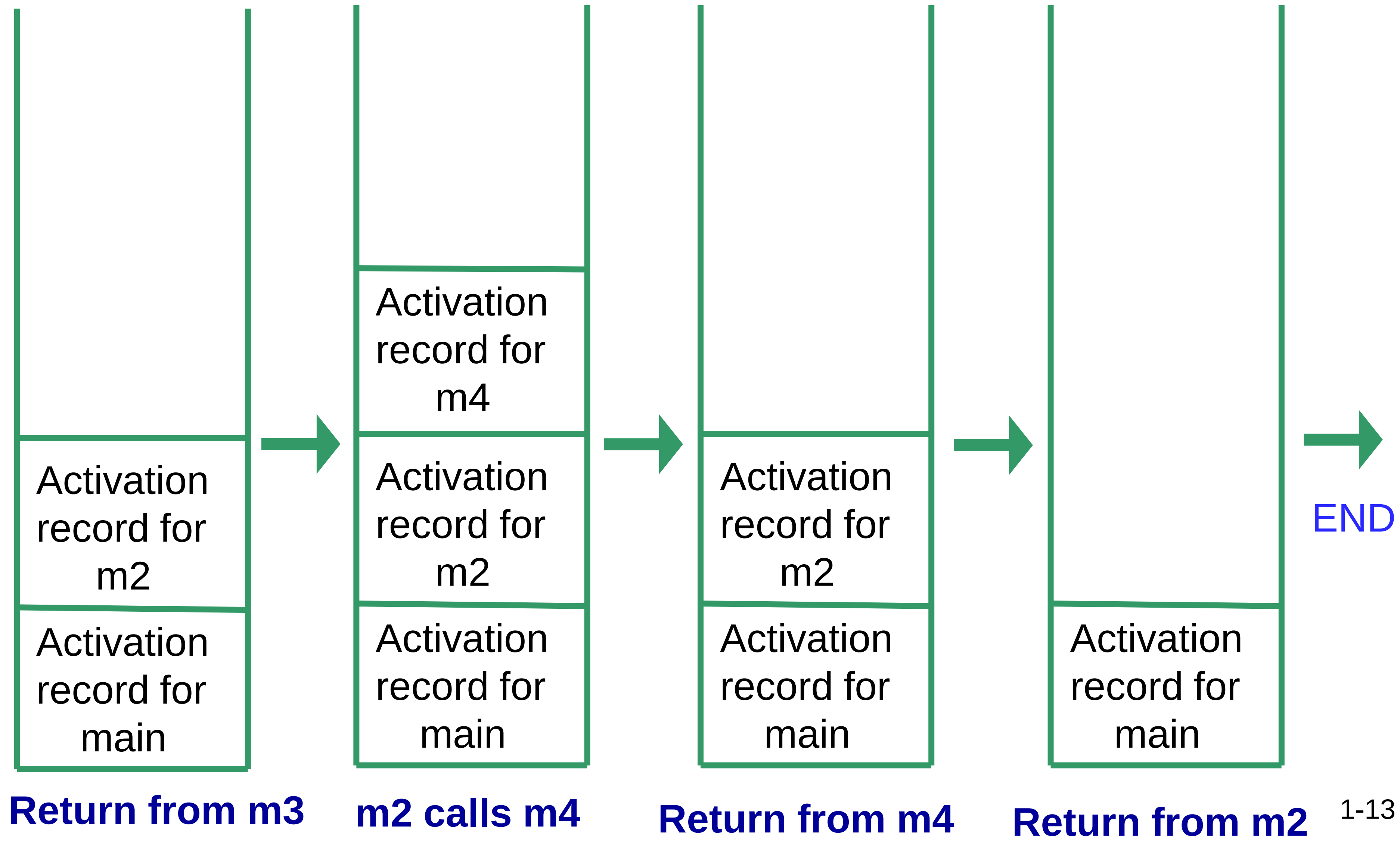
```
public static void m2( ) {
    System.out.println("Starting m2");
    System.out.println("m2 calling m3");
    m3();
    System.out.println("m2 calling m4");
    m4();
    System.out.println("Leaving m2");
    return;
}

public static void m3( ) {
    System.out.println("Starting m3");
    System.out.println("Leaving m3");
    return;
}
```

Execution Stack for Execution of above Program



Execution Stack for Execution of above Program



Execution of the Program

- When the **main** method is invoked:
    - An activation record for main is created and pushed onto the execution stack
- When **main** calls the method **m2**:
    - An activation record for m2 is created and pushed onto the execution stack
- When **m2** calls **m3**:
    - An activation record for m3 is created and pushed onto the execution stack
- When **m3** terminates, its activation record is popped off and control returns to **m2**

Execution of the Program

- When **m2** next calls **m4**:
  - What happens next?
  - What happens when **m4** terminates?

- What happens when **m2** terminates?

- What happens when **main** terminates?

  Its activation record is popped off and control returns to the operating system

Activation Records

- We will now look at an examples of what is in the activation record for a method with
  - primitive type variables, and
  - non-primitive variables

Activation Records

```
public static int m2 (int param2) {
    int local2 = 1;
    Integer i = new Integer(3);
    m3 (5);
    return local2 + param2 + m4(3);
}
public static void m3 (int param3) {
    int[] arr = new int[param3];
}
public static int m4 (int param4) {
    return param4 * 2;
}
public static void main (String[] args) {
    int local1 = m2(5);
}
```

We will show how this program is executed to explain how activation records are used and how they allow methods to be invoked.

Activation Records

```
public static int m2 (int param2) {
    int local2 = 1;
    Integer i = new Integer(3);
    m3 (5);
    return local2 + param2 + m4(3);
}
public static void m3 (int param3) {
    int[] arr = new int[param3];
}
public static int m4 (int param4) {
    return param4 * 2;
}
public static void main (String[] args) {
    int local1 = m2(5);
}
```

To execute this program, it is first compiled and translated to Java bytecode and stored in the static heap.

```
01001100011101001011010111
01001010110101010001111011
10101001010101000111001011
11101011101011101001110000
11010101010000100011100011
10111010011000111011000001
11010100100010101000101011
```

Static heap

Activation Records

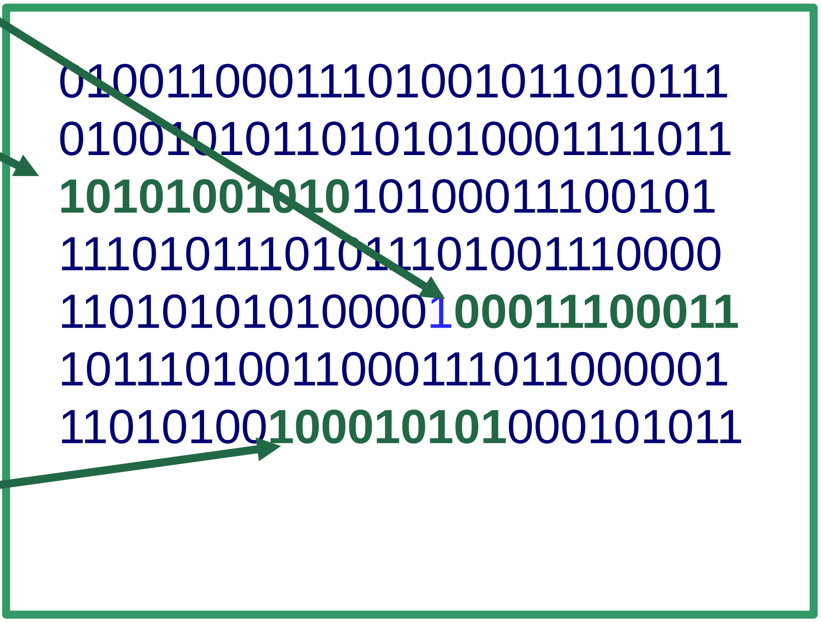
```
       public static int m2 (int param2) {
           int local2 = 1;
           Integer i = new Integer(3);
addr2      m3 (5);
addr3      return local2 + param2 + m4(3);
       }

       public static void m3 (int param3) {
           int[] arr = new int[param3];
       }

       public static int m4 (int param4) {
           return param4 * 2;
       }

       public static void main (String[] args) {
addr1      int local1 = m2(5);
       }
```

Each instruction is assigned an address. We indicated the addresses of method invocations in the program



```
01001100011101001011010111
01001010110101010001111011
10101001010101000111100101
11101011101011101001110000
11010101010000100011100011
10111010011000111011000001
11010100100010101000101011
```

Static heap

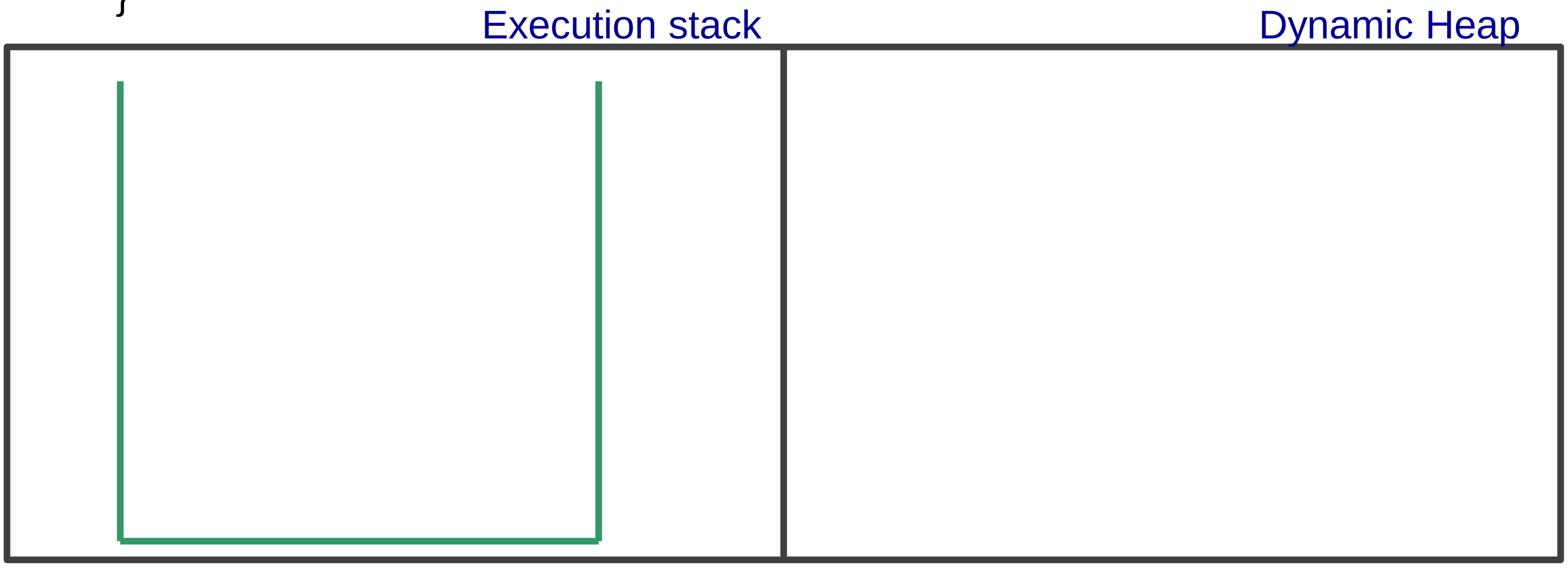
Activation Records

```
      public static int m2 (int param2) {
          int local2 = 1;
          Integer i = new Integer(3);
addr2     m3 (5);
addr3     return local2 + param2 + m4(3);
      }

      public static void m3 (int param3) {
          int[] arr = new int[param3];
      }
```

```
public static int m4 (int param4) {
    return param4 * 2;
}

public static void main (String[] args) {
    int local1 = m2(5);  addr1
}
```



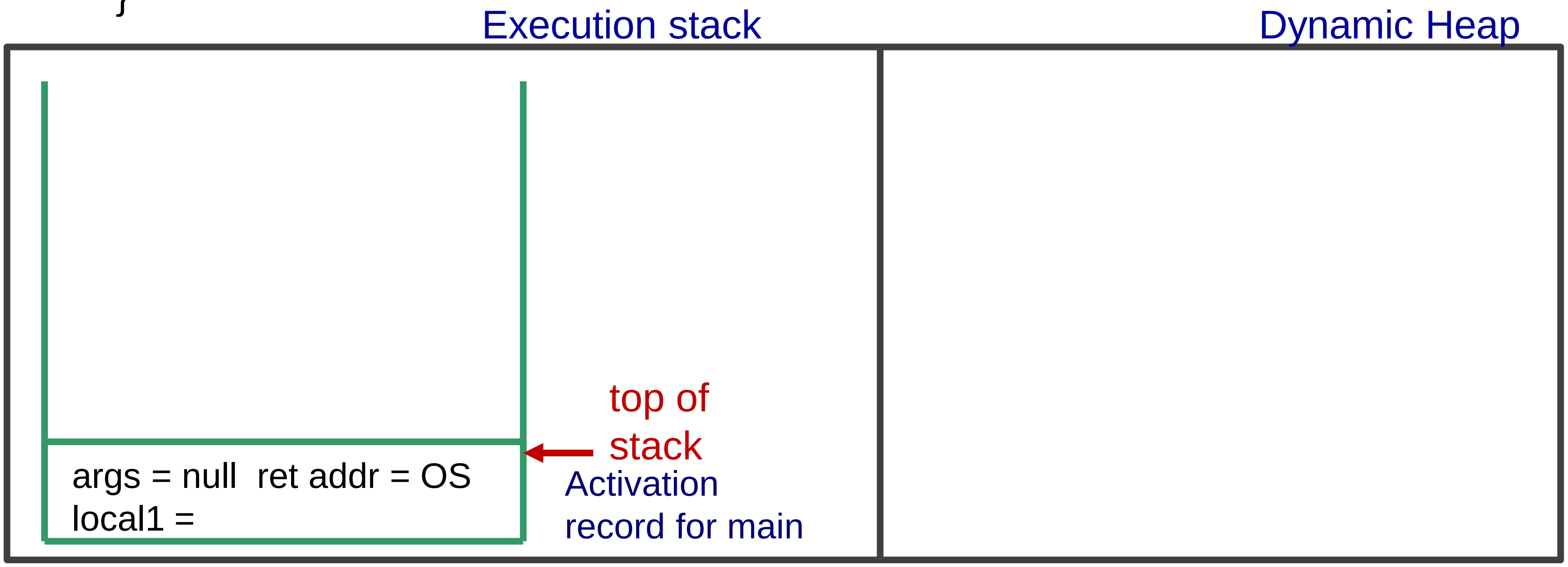
Activation Records

```
    public static int m2 (int param2) {
        int local2 = 1;
        Integer i = new Integer(3);
addr2   m3 (5);
addr3   return local2 + param2 + m4(3);
    }

    public static void m3 (int param3) {
        int[] arr = new int[param3];
    }
```

```
public static int m4 (int param4) {
    return param4 * 2;
}

public static void main (String[] args) {
    int local1 = m2(5);  addr1
}
```



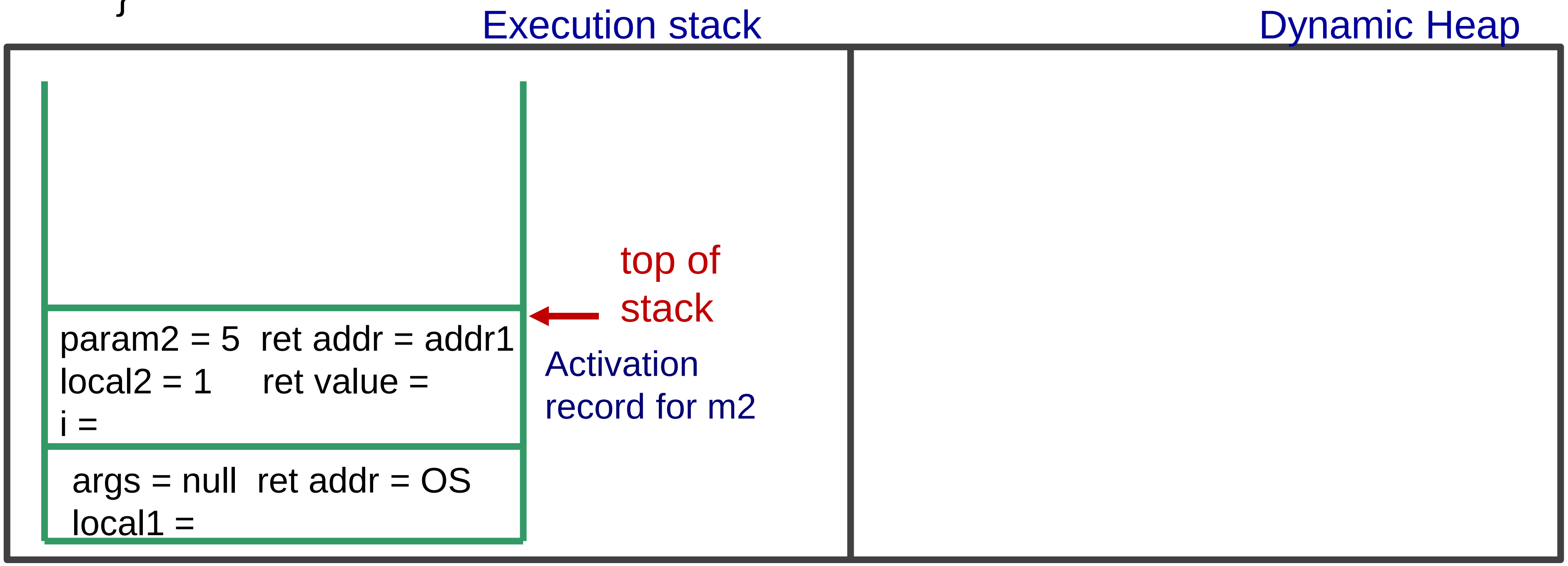
Activation Records

```
        public static int m2 (int param2) {
            int local2 = 1;
            Integer i = new Integer(3);
addr2       m3 (5);
addr3       return local2 + param2 + m4(3);
        }

        public static void m3 (int param3) {
            int[] arr = new int[param3];
        }

public static int m4 (int param4) {
    return param4 * 2;
}

public static void main (String[] args) {
    int local1 = m2(5);   addr1
}
```



```
param2 = 5   ret addr = addr1
local2 = 1   ret value =
i =

args = null  ret addr = OS
local1 =
```

Activation Records

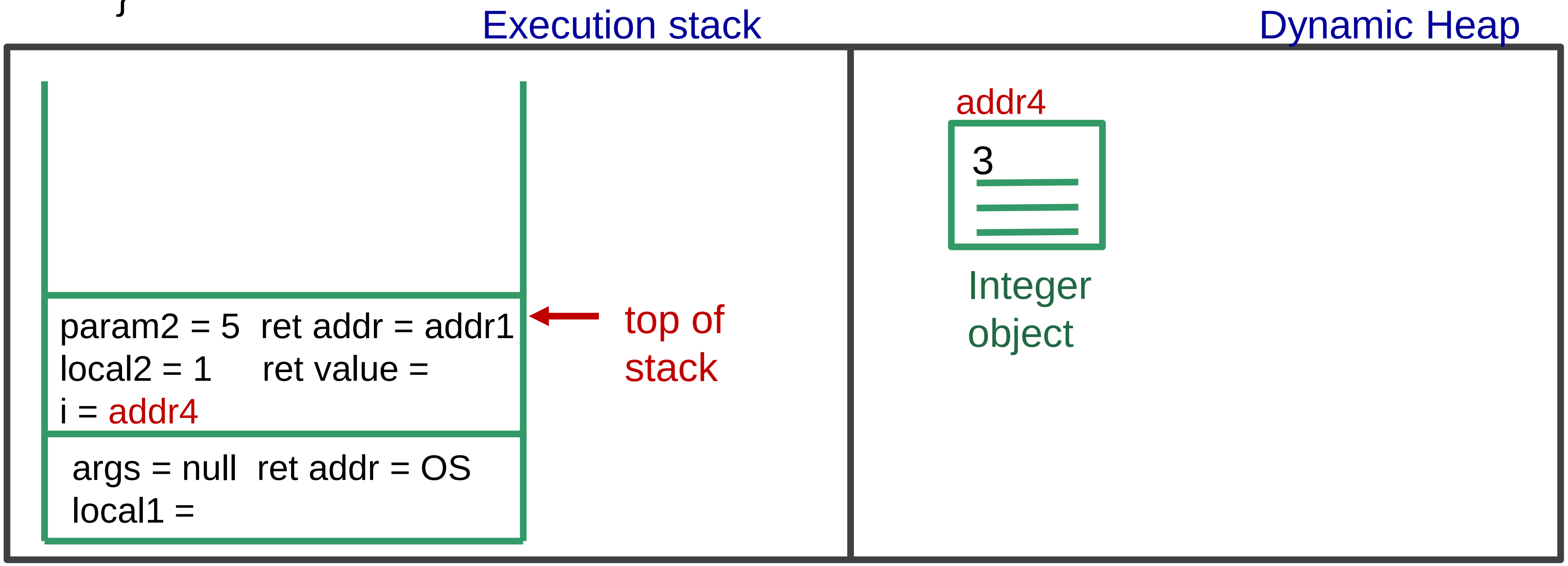
```
public static int m2 (int param2) {
    int local2 = 1;
    Integer i = new Integer(3);
addr2 m3 (5);
addr3 return local2 + param2 + m4(3);
}

public static void m3 (int param3) {
    int[] arr = new int[param3];
}
```

```
public static int m4 (int param4) {
    return param4 * 2;
}

public static void main (String[] args) {
    int local1 = m2(5);  addr1
}
```

An object is created



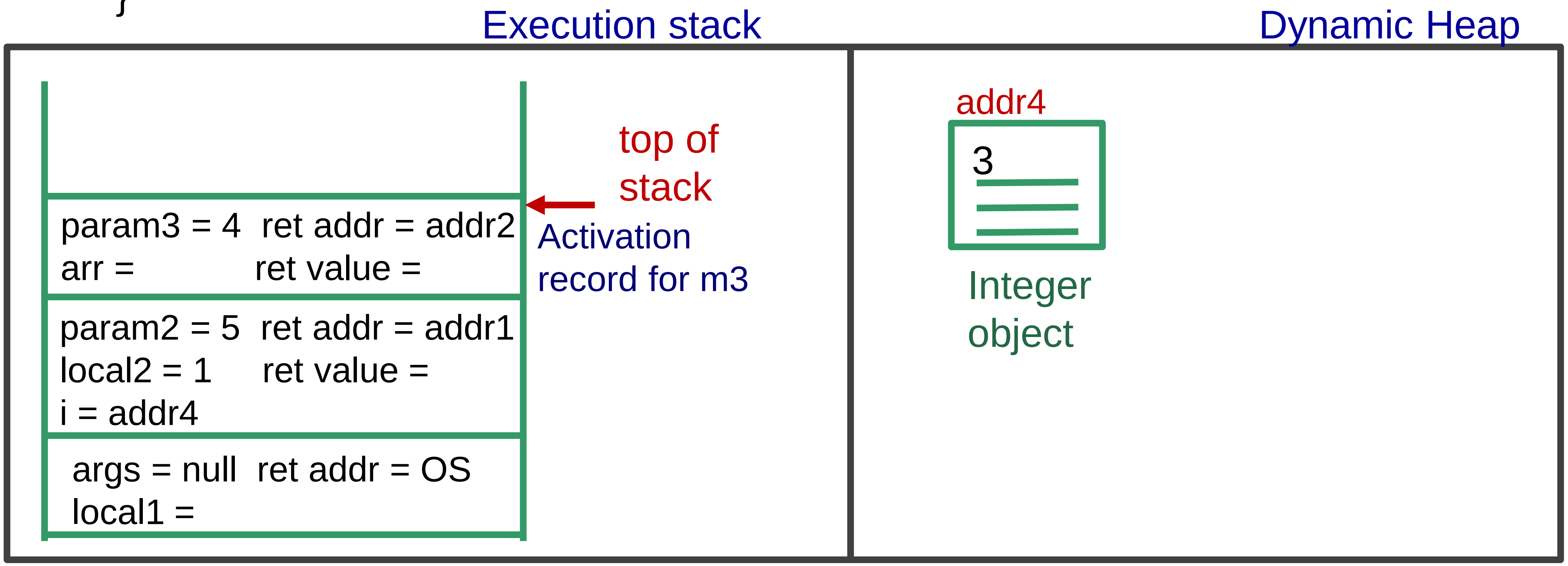
Activation Records

```
      public static int m2 (int param2) {
          int local2 = 1;
          Integer i = new Integer(3);
addr2     m3 (4);
addr3     return local2 + param2 + m4(3);
      }

      public static void m3 (int param3) {
          int[] arr = new int[param3];
      }
```

```
public static int m4 (int param4) {
    return param4 * 2;
}

public static void main (String[] args) {
    int local1 = m2(5);   addr1
}
```



Activation Records

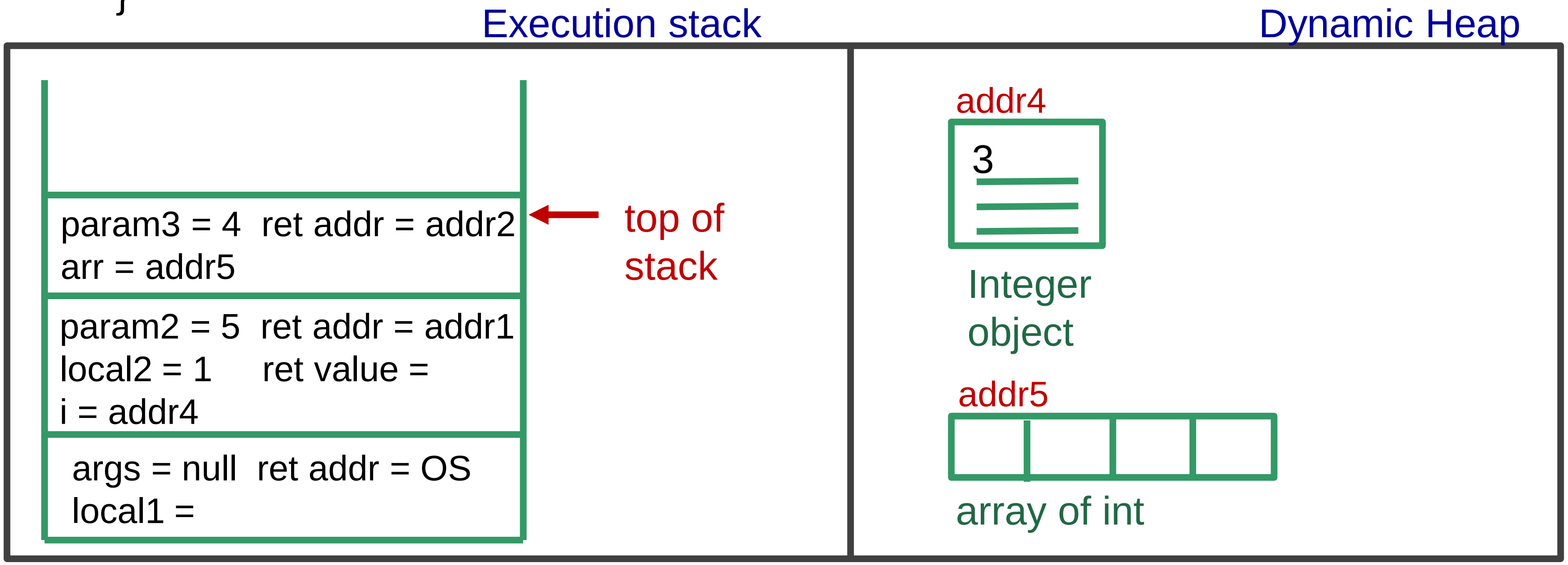
```
public static int m2 (int param2) {
    int local2 = 1;
    Integer i = new Integer(3);
addr2 m3 (4);
addr3 return local2 + param2 + m4(3);
}

public static void m3 (int param3) {
    int[] arr = new int[param3];
}
```

```
public static int m4 (int param4) {
    return param4 * 2;
}

public static void main (String[] args) {
    int local1 = m2(5);   addr1
}
```

An object is created



Activation Records

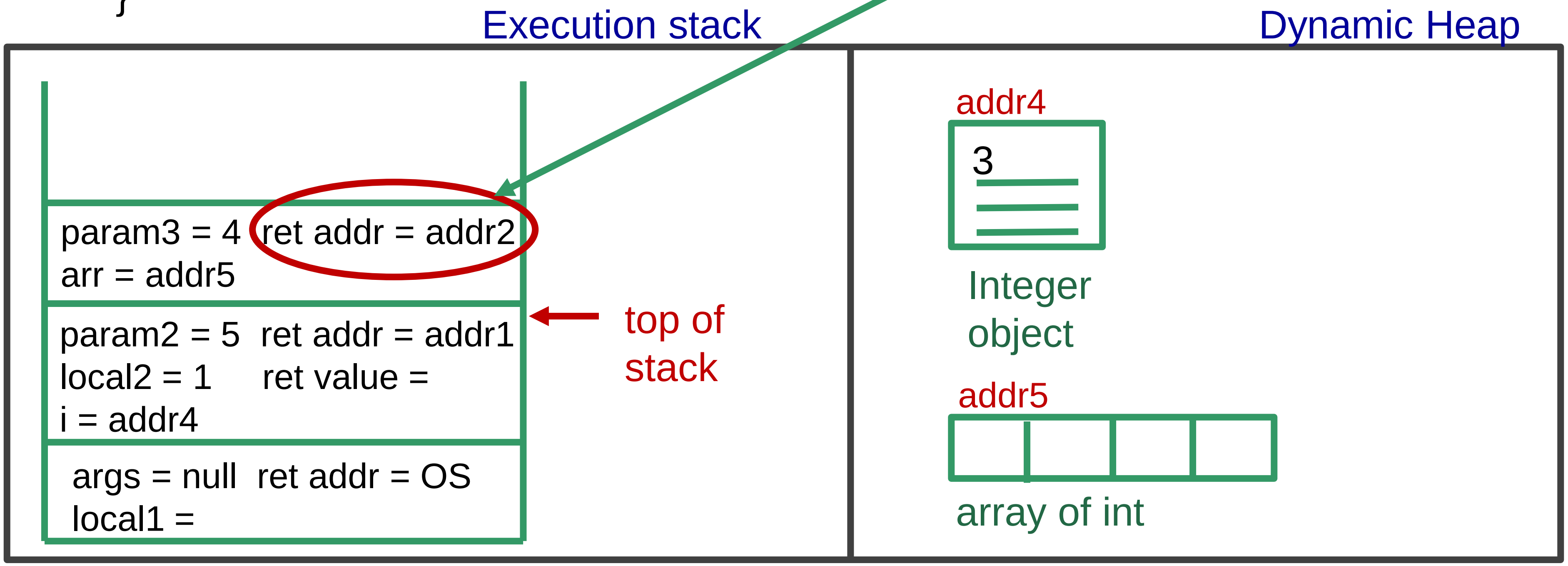
```
    public static int m2 (int param2) {
        int local2 = 1;
        Integer i = new Integer(3);
addr2   m3 (4);
addr3   return local2 + param2 + m4(3);
    }

    public static void m3 (int param3) {
        int[] arr = new int[param3];
    }

public static int m4 (int param4) {
    return param4 * 2;
}

public static void main (String[] args) {
    int local1 = m2(5);  addr1
}
```

Execution continues after addr2



Activation Records

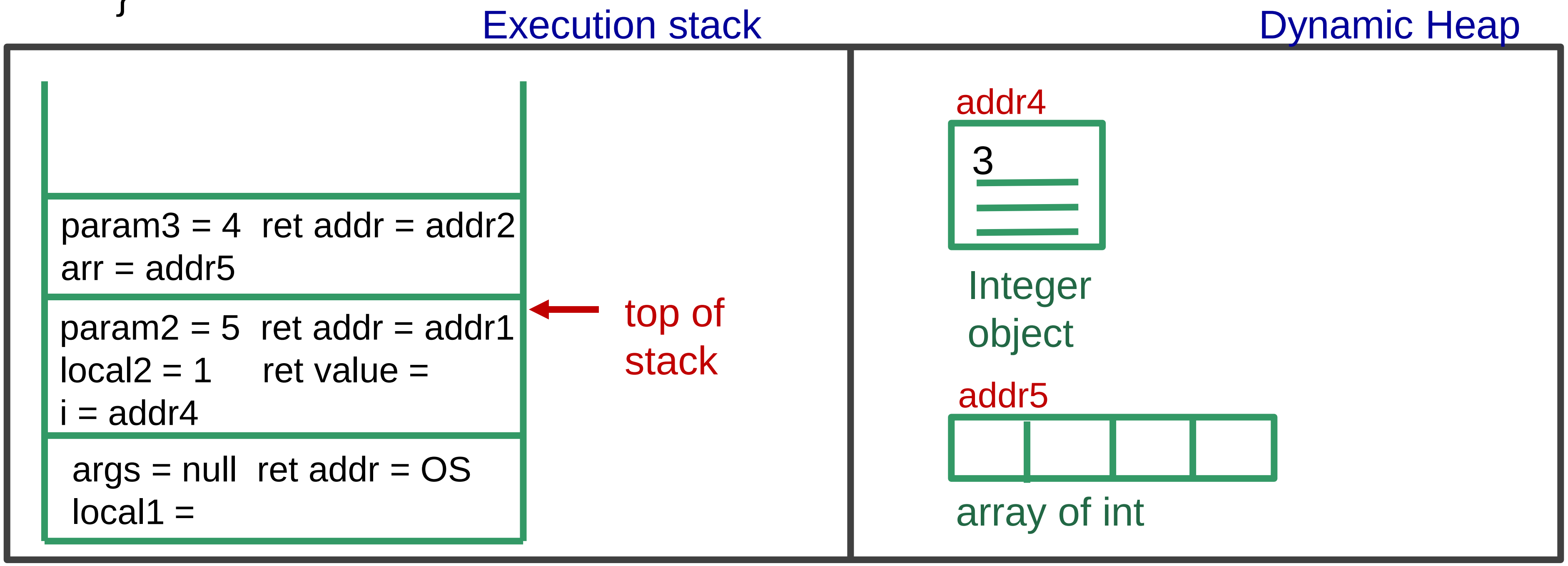
```
       public static int m2 (int param2) {
           int local2 = 1;
           Integer i = new Integer(3);
addr2      m3 (4);
addr3      return local2 + param2 + m4(3);
       }

       public static void m3 (int param3) {
           int[] arr = new int[param3];
       }
```

```
public static int m4 (int param4) {
    return param4 * 2;
}

public static void main (String[] args) {
    int local1 = m2(5);   addr1
}
```

This invocation to m4 is next made



Activation Records

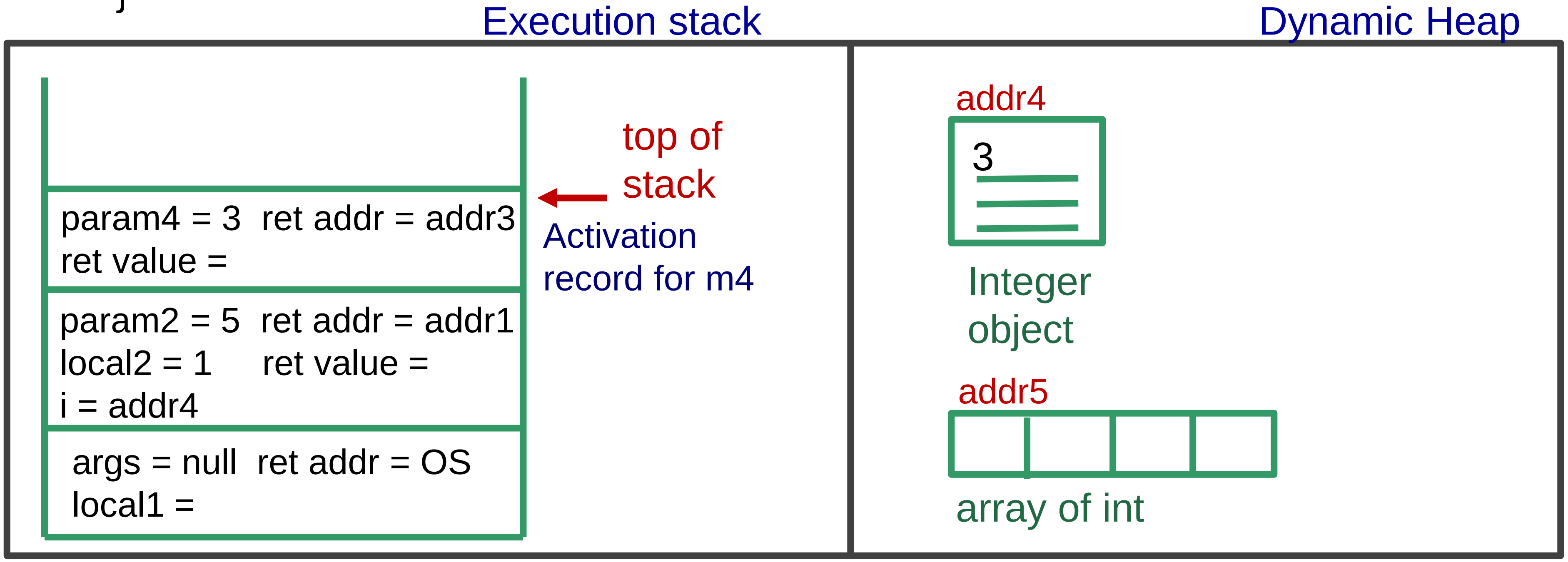
```
     public static int m2 (int param2) {
         int local2 = 1;
         Integer i = new Integer(3);
addr2    m3 (4);
addr3    return local2 + param2 + m4(3);
     }

     public static void m3 (int param3) {
         int[] arr = new int[param3];
     }

public static int m4 (int param4) {
    return param4 * 2;
}

public static void main (String[] args) {
    int local1 = m2(5);  addr1
}
```

This invocation to m4 is next made



Activation Records

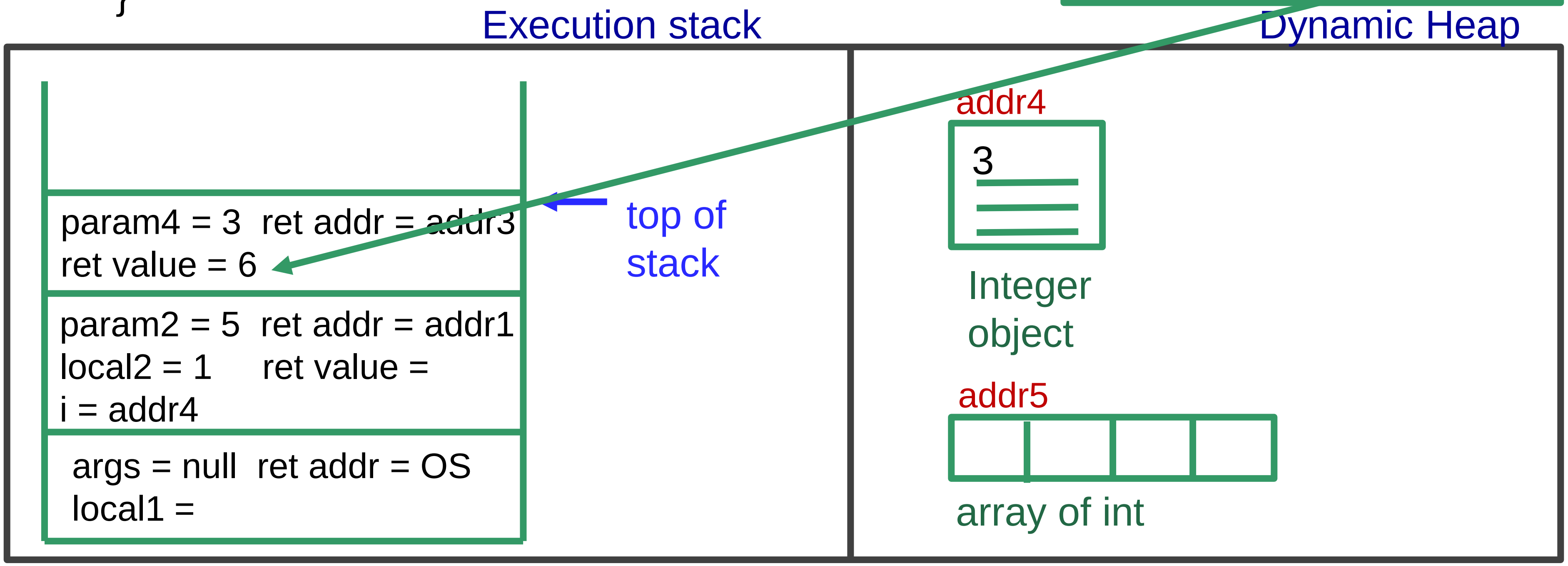
```
public static int m2 (int param2) {
    int local2 = 1;
    Integer i = new Integer(3);
addr2 m3 (4);
addr3 return local2 + param2 + m4(3);
}

public static void m3 (int param3) {
    int[] arr = new int[param3];
}

public static int m4 (int param4) {
    return param4 * 2;
}

public static void main (String[] args) {
    int local1 = m2(5);  addr1
}
```

Return value is computed and stored in activation record



Activation Records

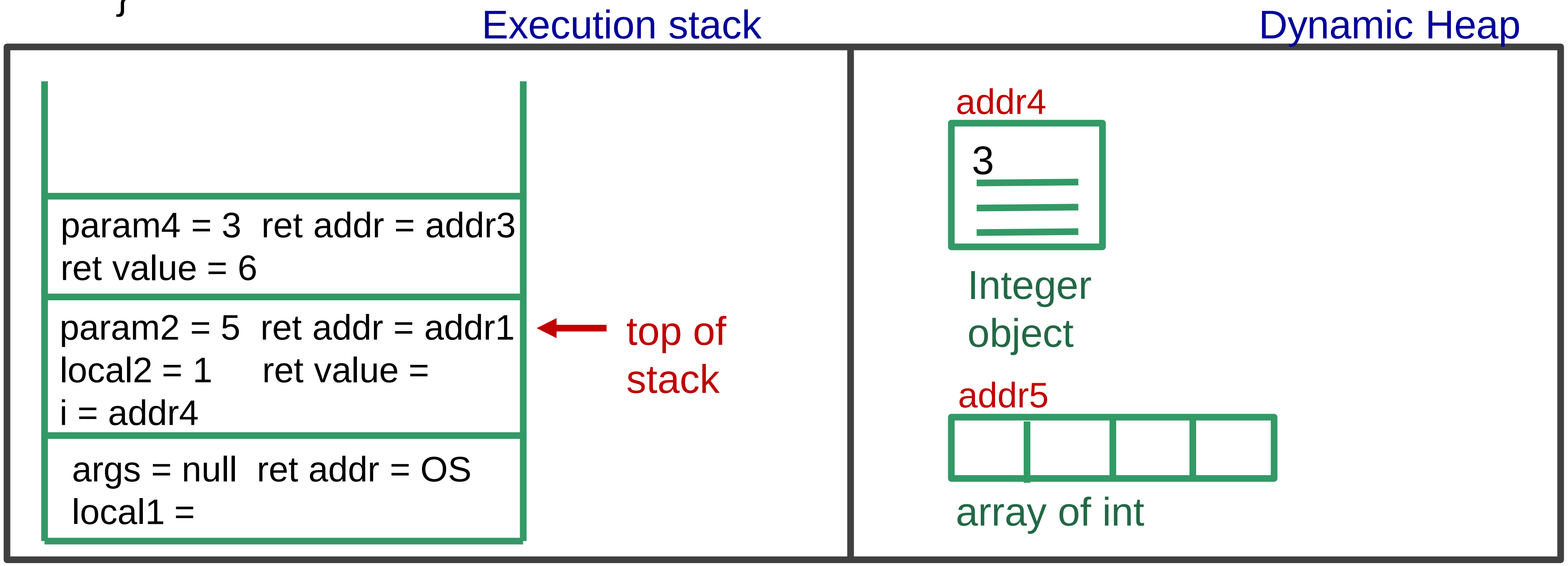
```
    public static int m2 (int param2) {
        int local2 = 1;
        Integer i = new Integer(3);
addr2   m3 (4);
addr3   return local2 + param2 + m4(3);
    }

    public static void m3 (int param3) {
        int[] arr = new int[param3];
    }
```

```
public static int m4 (int param4) {
    return param4 * 2;
}

public static void main (String[] args) {
    int local1 = m2(5);  addr1
}
```

Execution continues after addr3



Activation Records

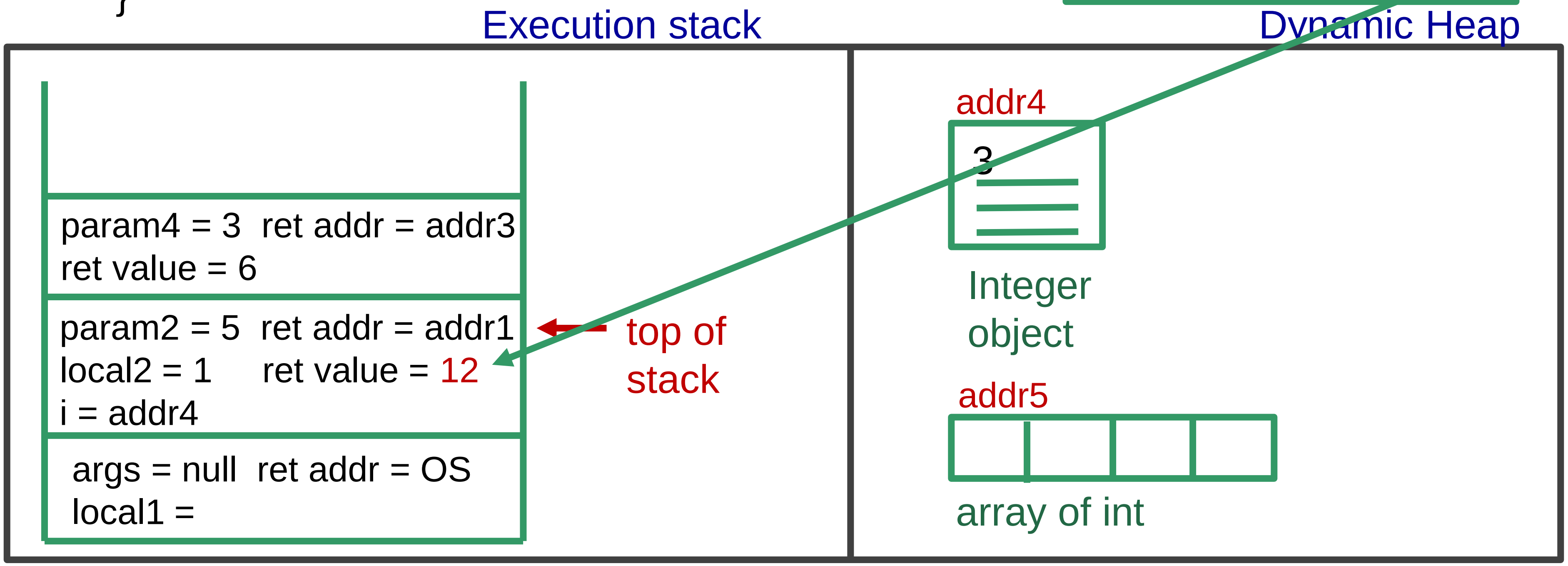
```
      public static int m2 (int param2) {
          int local2 = 1;
          Integer i = new Integer(3);
addr2 m3 (4);
addr3 return local2 + param2 + m4(3);
      }

      public static void m3 (int param3) {
          int[] arr = new int[param3];
      }
```

```
public static int m4 (int param4) {
    return param4 * 2;
}

public static void main (String[] args) {
    int local1 = m2(5);  addr1
}
```

Return value is computed and store in activation record



Activation Records

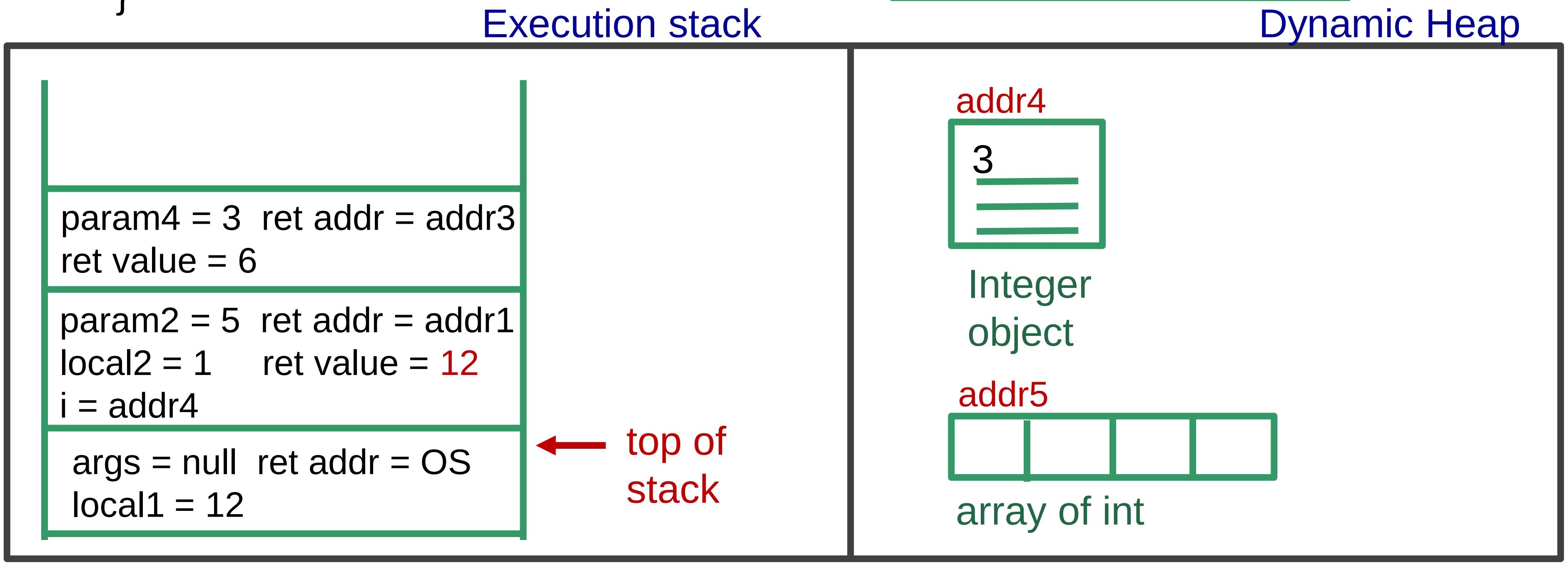
```
       public static int m2 (int param2) {
           int local2 = 1;
           Integer i = new Integer(3);
addr2      m3 (4);
addr3      return local2 + param2 + m4(3);
       }

       public static void m3 (int param3) {
           int[] arr = new int[param3];
       }
```

```
public static int m4 (int param4) {
    return param4 * 2;
}

public static void main (String[] args) {
    int local1 = m2(5);  addr1
}
```

Execution continues after addr1



Activation Records

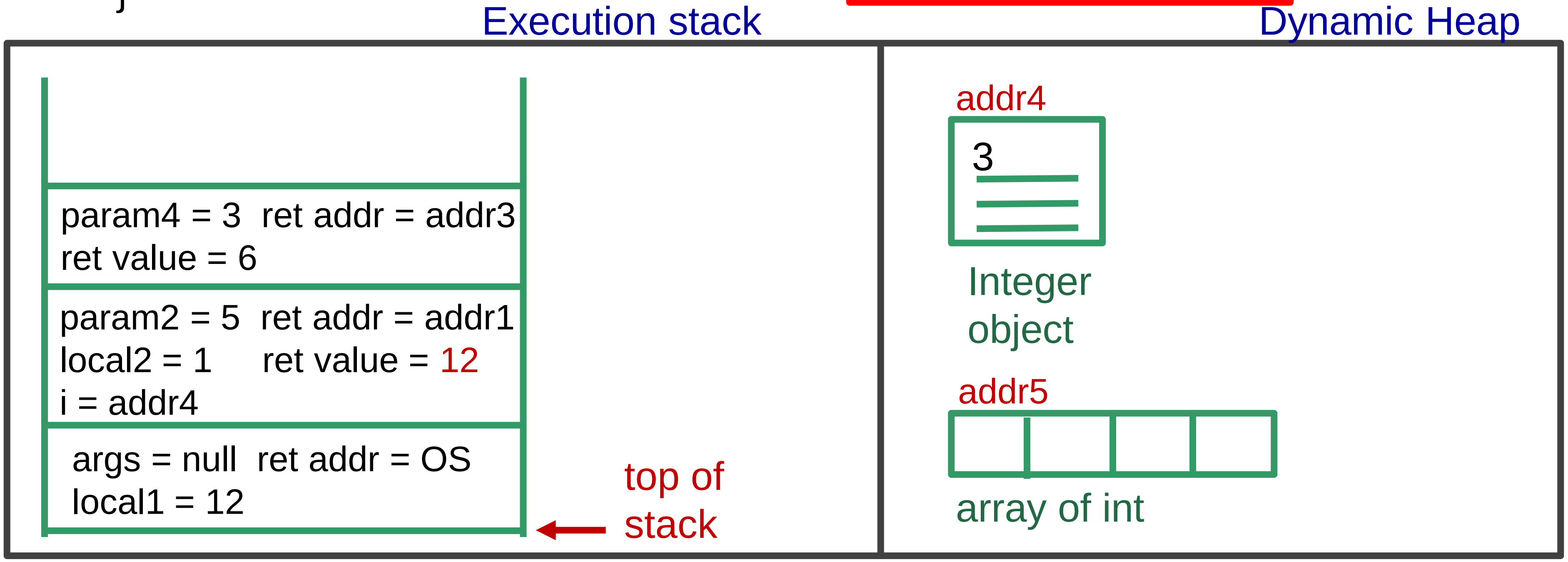
```
    public static int m2 (int param2) {
        int local2 = 1;
        Integer i = new Integer(3);
addr2   m3 (4);
addr3   return local2 + param2 + m4(3);
    }

    public static void m3 (int param3) {
        int[] arr = new int[param3];
    }
```

```
public static int m4 (int param4) {
    return param4 * 2;
}

public static void main (String[] args) {
    int local1 = m2(5);  addr1
}
```

Program ends and control goes back to Operating System



Activation Records – Example 2

```
public class CallFrameDemo2 {
    private static void printAll (String s1, String s2, String s3) {
            System.out.println(s1.toString( ));
            System.out.println(s2.toString( ));
            System.out.println(s3.toString( ));
     }
    public static void main (String args[ ]) {
            String str1, str2, str3;

            str1 = new String(" string 1 ");
            str2 = new String(" string 2 ");
            str3 = new String(" string 3 ");

            printAll(str1, str2, str3);
     }
}
```

Activation Records – Example 2

Draw a picture of the execution stack and of the heap as the above program executes:

- Activation record for main
- Activation record for String constructor for str1 – then popped off
- Activation record for String constructor for str2 – then popped off
- Activation record for String constructor for str3 – then popped off
- Activation record for printAll
- Activation record for toString for str1 – then popped off
- Activation record for System.out.println – then popped off
- etc.

Activation Records – Example 2

- What will be stored in the activation record for **main**?
  - Address to return to operating system
  - Variable **args**
  - Variable **str1**
    - Initial value?
    - Value after return from String constructor?
  - Variable **str2**
  - Variable **str3**

- What will be in the activation record for **printAll**?

Memory Deallocation

- What happens when a method returns?
  - On the execution stack:
    - The activation record is popped off when the method returns
    - So, that memory is deallocated

Memory Deallocation

- What happens to objects on the heap?
  - An object stays in the heap even if there is no longer a variable referencing it!
  - So, Java has automatic garbage collection
    - When memory is running low, objects which no longer have a variable referencing them are identified and their memory is deallocated